المملكة المغربية
Royaume du Maroc

المدرسة المحمدية للأشغال العمومية

RAPPORT DU PROJET

Elaboré par :

BENCHARI Khaoula

CHOUKRI Hind

Encadré par :

Mme ADDOU

1GI

28/04/2019

TABLES DE MATIERES

I. Introduction

1.1 Contexte

Ce projet s'inscrit dans le cadre de module « Résolution des problème ». L'objectif est de programmer le jeu du moulin en langage C/C++, en utilisant une intelligence artificielle IA sophistiquée en utilisant nos connaissances acquises pendant ce module.

1.2 Présentation du jeu

Le jeu du moulin est un jeu de société traditionnel européen qui trace ses origines de l'Egypte antique et de la Rome antique et qui se joue sur un plateau 7x7.

1.3 Travail réalisé

Dans un premier temps, on présente les versions console et graphique réalisées en langage C. Ensuite, on va présenter les différents algorithmes, structures et fonctions utilisés pour réaliser le travail demandé, tout en décrivant leurs rôles et leur fonctionnement global.

1.4 Outils utilisés

- **Code Blocks** : implémenter le jeu du moulin ainsi que les stratégies **MiniMax** et **Alpha-Beta**.
- **Adobe Photoshop** : travailler sur les images utilisées dans l'interface graphique.
- **La bibliothèque SDL (Simple DirectMedia Layer)** : réaliser une interface graphique en travaillant avec la bibliothèque **SDL_image** pour insérer les images et la bibliothèque **SDL-ttf** pour écrire du texte.
- **Sublime Text , Xcode** : Réaliser une bonne mise en forme du rapport .

II. Modélisation du problème

1.1 Simulation

- **Espace d'états :**

```
typedef struct Etat
{
 int tableau[8][8];
 int jeton_b,jeton_r;
 int j_b_pose,j_r_pose;

}Etat;
```

L'espace d'états sera représenté par :

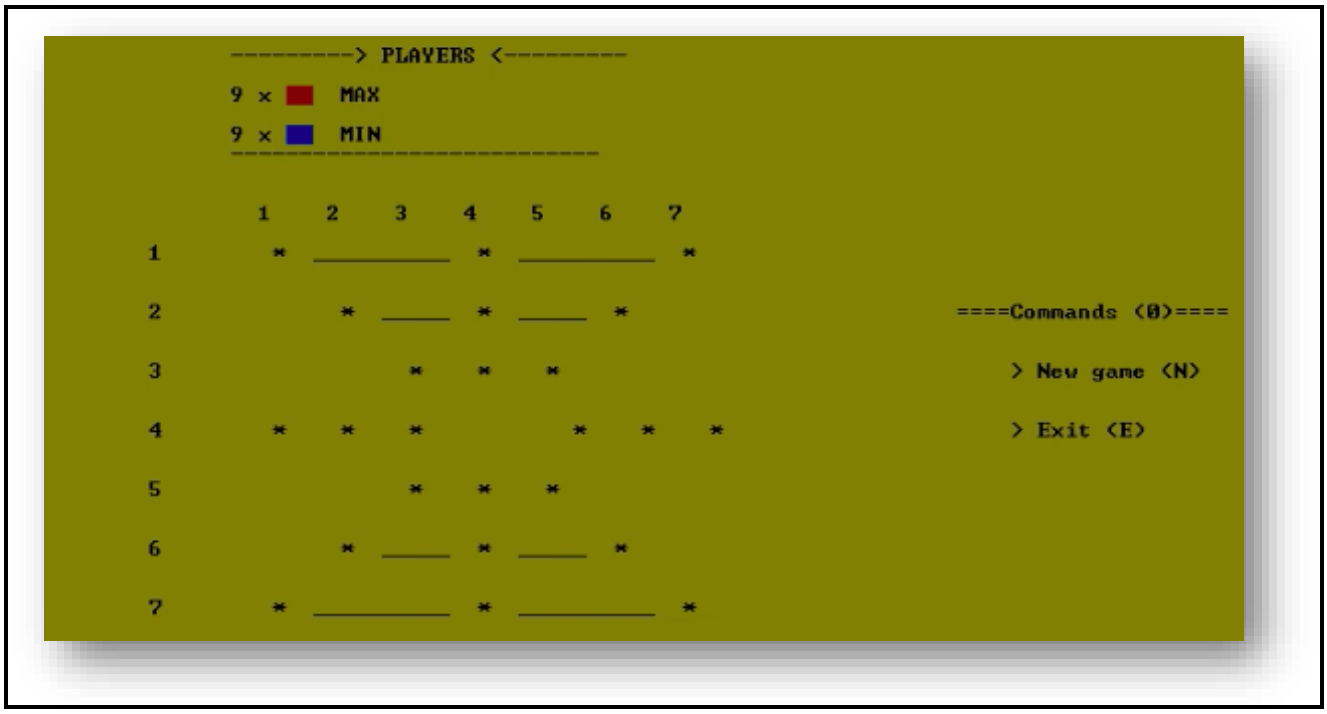
- tableau[i][j] : tel que pour : i ∈ [1,7] et j ∈ [1,7] représentent les intersections du plateau ainsi que i=0 et j=0 sont réservés pour le choix entre : New game ou Exit .

```
====Commands (0)====

    > New game (N)

    > Exit (E)
```

- Jeton_r : représente les jetons rouges dont possède le joueur Max au début .
- Jeton_b : représente les jetons bleus dont possède le joueur Min au début .
- J_r_pose : représente les jetons rouges posées par le joueur Max sur le plateau .
- J_b_pose : représente les jetons bleus posées par le joueur Min sur le plateau .

- **Etat initial :**

Le jeu commence avec un plateau vide entre deux joueurs qui possède chacun d'eux 9 pions (jetons).



Alors l'état initial sera représenté à l'aide de la fonction **void f_etatinitial (Etat *E)** :

- tableau[8][8] : est initialisé selon les lignes et les colonnes tels que si le point représente une intersection vide du plateau donc celui-ci sera initialisé par la valeur 0, sinon on l'affecte la valeur 2.

Tableau [1][j] = 0 , j ∈ {1,4,7 } : les points d'intersections de la 1ére ligne sont vides
Tableau[1][j] = 2 , j ∈ {2,3,5,6} : on ne peut pas poser aucun jeton dans ces points

Tableau [2][j] = 0 , j ∈ {2,4,6 } : les points d'intersections de la 2éme ligne sont vides
Tableau[2][j] = 2 , j ∈ {1,3,5,7} : on ne peut pas poser aucun jeton dans ces points

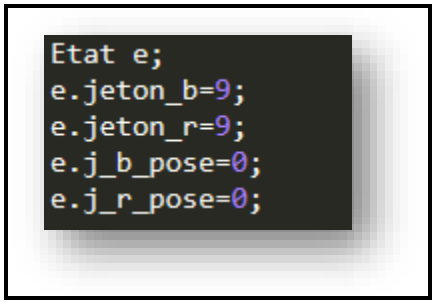
Tableau [3][j] = 0 , j ∈ {3,4,5 } : les points d'intersections de la 3ème ligne sont vides
Tableau[3][j] = 2 , j ∈ {1,2,6,7} : on ne peut pas poser aucun jeton dans ces points

…

```
for(i=0;i<8;i++)
        for(j=0;j<8;j++)
            e.tableau[i][j]=0;
```

```
for(j=1;j<=7;j++)
 {
   if(j!=1&&j!=4&&j!=7)
   {
       e.tableau[1][j]=2;
       e.tableau[7][j]=2;
   }
   if(j!=2&&j!=4&&j!=6)
   {
       e.tableau[2][j]=2;
       e.tableau[6][j]=2;
   }
   if(j!=3&&j!=4&&j!=5)
   {
       e.tableau[3][j]=2;
       e.tableau[5][j]=2;
```

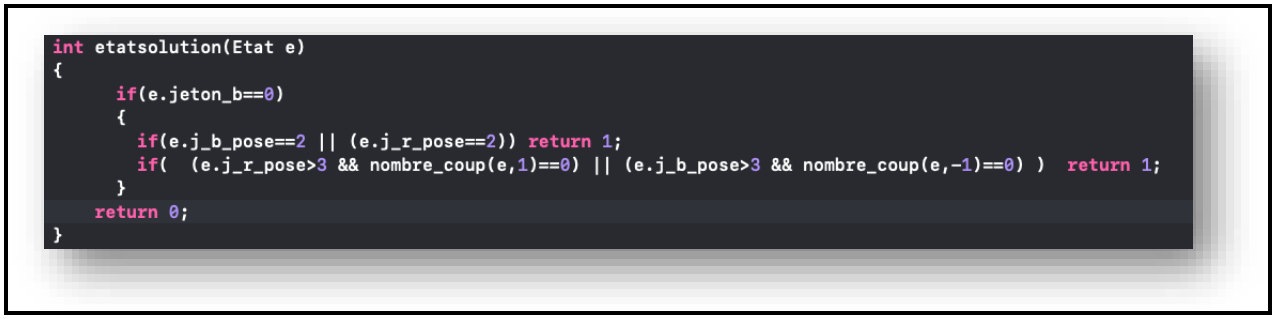
- Jeton_r = 9
- Jeton_b = 9
- J_r_pose = 0
- J_b_pose = 0



```
Etat e;
e.jeton_b=9;
e.jeton_r=9;
e.j_b_pose=0;
e.j_r_pose=0;
```

- **Etat final :**

Pour l'état final on a deux cas pour chaque joueur : s'il n'a que deux jetons ou s'il est bloqué (il n'a aucun coup à effectuer).



```
int etatsolution(Etat e)
{
      if(e.jeton_b==0)
      {
        if(e.j_b_pose==2 || (e.j_r_pose==2)) return 1;
        if(  (e.j_r_pose>3 && nombre_coup(e,1)==0) || (e.j_b_pose>3 && nombre_coup(e,-1)==0) )  return 1;
      }
    return 0;
}
```

- **Règles de production :**

Il existe en génèral quatres règles :

- R1 (*pose*) : poser un jeton sur une intersection vide du plateau .

```
liste* poser(Etat etatcourant,int maxplayer, liste *1 )
{
    int i,j;
    for(i=1;i<8;i++)
                    for(j=1;j<8;j++)
                        if(etatcourant.tableau[i][j]==0)
                            {
                                etatcourant.tableau[i][j]=maxplayer;
                                if(est_un_moulin(i,j,maxplayer,etatcourant)==1) l=enlevercase(etatcourant,maxplayer,l);
                                 else    inserer(etatcourant,&l);
                                etatcourant.tableau[i][j]=0;
                            }
    return l;
}
```

- R2 (*mouvement*) : glisser les jetons d'un point d' intersection à un autre de la ligne, quand ils sont tous posés (mouvement à la fois).

```
liste* glisser(Etat etatcourant,int maxplayer, liste *1 )
{
    int i,j,k,m;
    for(i=1;i<8;i++)
                    for(j=1;j<8;j++)
                        if(etatcourant.tableau[i][j]==maxplayer)
                            {
                                for(k=1;k<8;k++)
                                    for(m=1;m<8;m++)
                                        if(etatcourant.tableau[k][m]==0 && test_glisser(i,j,k,m)==1)
                                            {
                                            etatcourant.tableau[i][j]=0;
                                            etatcourant.tableau[k][m]=maxplayer;
                                            if(est_un_moulin(k,m,maxplayer,etatcourant)==1) l=enlevercase(etatcourant,maxplayer,l);
                                            else inserer(etatcourant,&l);
                                             etatcourant.tableau[i][j]=maxplayer;
                                            etatcourant.tableau[k][m]=0;
                                            }


                            }


    return l;
}
```

- R3 (*saut*) : sauter d'un point d'intersection à un autre ,quand le joueur n'a que 3 jetons .

```
liste* sauter(Etat etatcourant,int maxplayer, liste *l )
{
    int i,j,k,m;
    for(i=1;i<8;i++)
                    for(j=1;j<8;j++)
                        if(etatcourant.tableau[i][j]==maxplayer)
                            {
                                for(k=1;k<8;k++)
                                    for(m=1;m<8;m++)
                                        if(etatcourant.tableau[k][m]==0)
                                            {
                                            etatcourant.tableau[i][j]=0;
                                            etatcourant.tableau[k][m]=maxplayer;
                                            if(est_un_moulin(k,m,maxplayer,etatcourant)==1) l=enlevercase(etatcourant,maxplayer,l);
                                             else  inserer(etatcourant,&l);
                                             etatcourant.tableau[i][j]=maxplayer;
                                            etatcourant.tableau[k][m]=0;
                                            }
                            }
return l;}
```

- R4(*retrait*) : retirer un jeton à l'adversaire quand le joueur forme un moulin .

```
liste* enlevercase(Etat etatcourant,int maxplayer,liste *l)
{
    int i,j,c=0;
     if(maxplayer==1) etatcourant.j_b_pose--;
    else etatcourant.j_r_pose--;
    for(i=1;i<8;i++)
        for(j=1;j<8;j++)
          if(etatcourant.tableau[i][j]==(-1)*maxplayer && est_un_moulin(i,j,(-1)*maxplayer,etatcourant)==0)
            {
                etatcourant.tableau[i][j]=0;
                inserer(etatcourant,&l);
                etatcourant.tableau[i][j]=(-1)*maxplayer;
                c=1;
            }
       if(c==0)
        {
            for(i=1;i<8;i++)
            for(j=1;j<8;j++)
          if(etatcourant.tableau[i][j]==(-1)*maxplayer )
            {
                etatcourant.tableau[i][j]=0;
                inserer(etatcourant,&l);
                etatcourant.tableau[i][j]=(-1)*maxplayer;
            }
        }
    return l;
}
```

- **Fonction heuristique h (nœud) :**

La fonction heuristique se base sur trois parties :

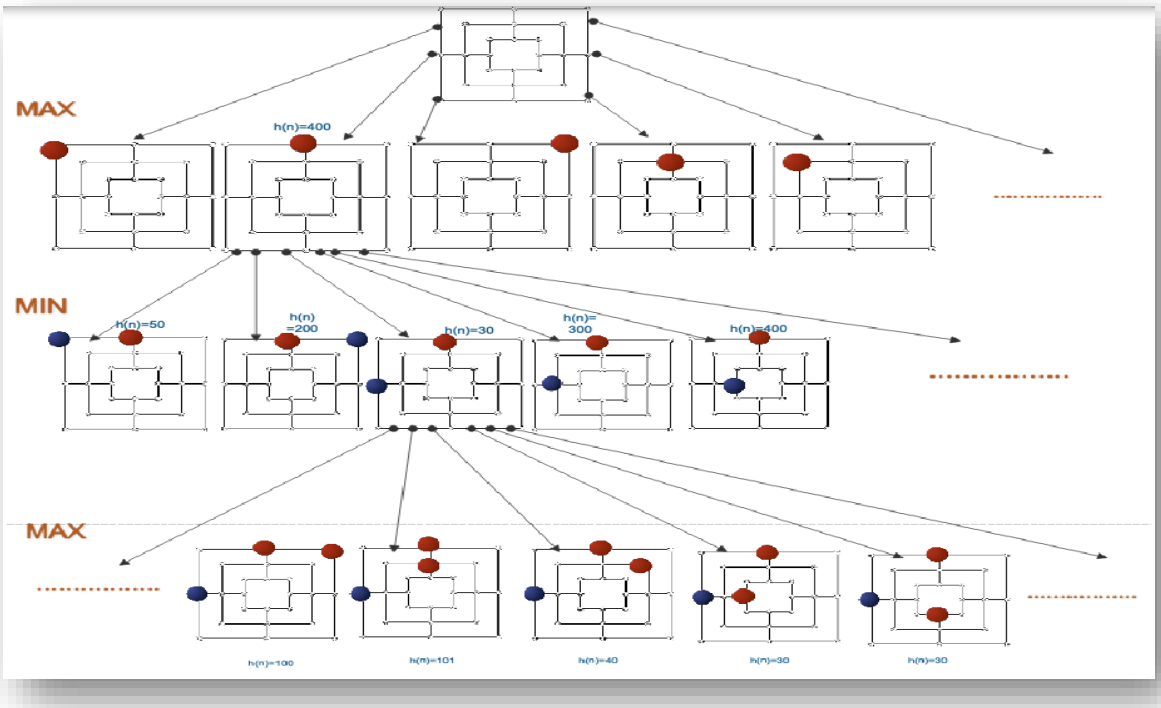
- Matériel : nombre de pièces restant pour chaque joueur.
- Liberté : nombre de coups possibles qu'un joueur peut effectuer.
- Moulins : nombre de moulins fermés.

Donc on a fait une fonction h qui est globale et fait appel aux deux autres fonctions.

```
int h(Etat);
int nombre_coup(Etat e,int maxplayer);
int nombre_moulin(Etat e, int maxplayer);
```

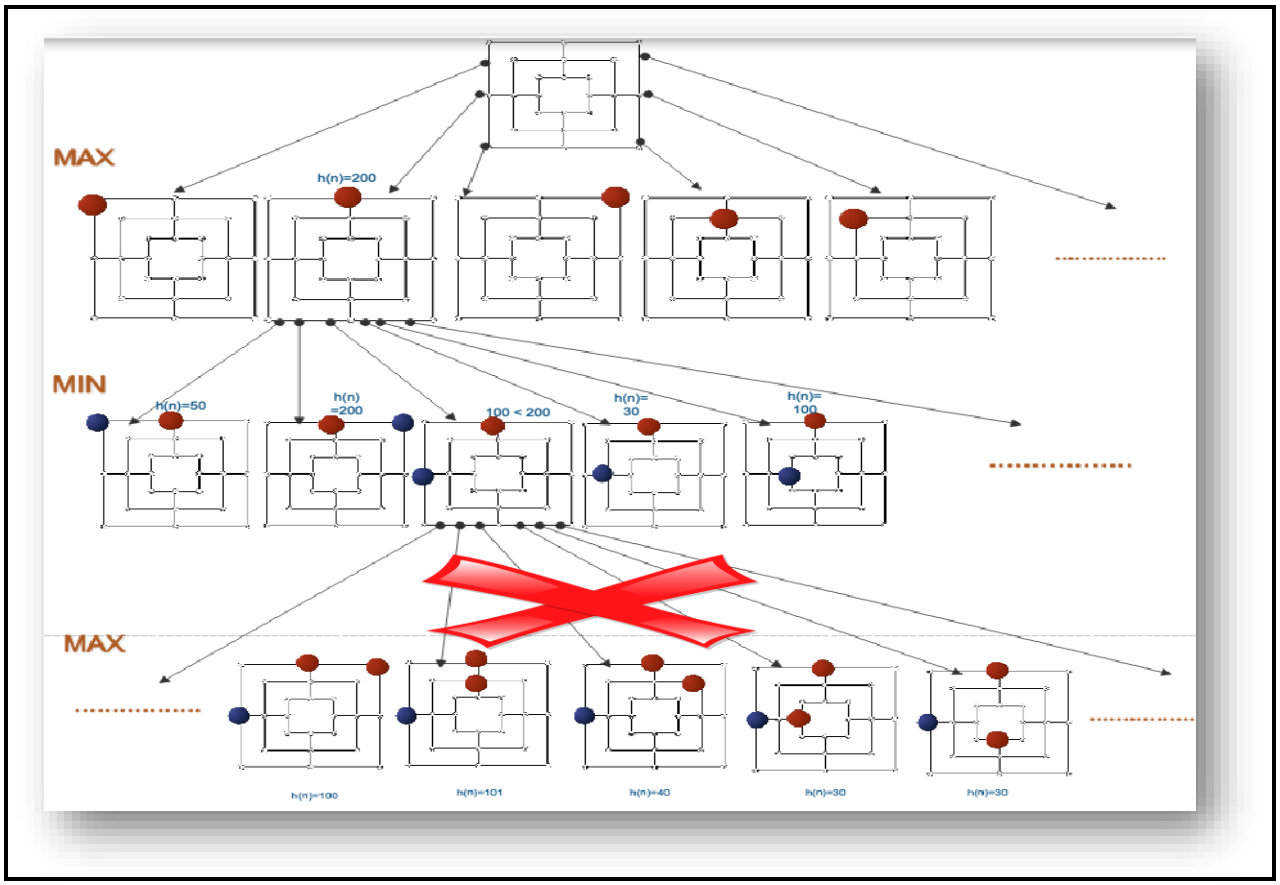
- **Représentation graphique :**
  - *Stratégie MiniMax :*

-Prévoir à l'avance les mouvements de l'adversaire.
- Prévoir un espace de recherche plus grand.
- Chercher le meilleur mouvement possible.



- *Stratégie Alpha-Beta*

  - Améliorer l'algorithme MiniMax.
  - Élaguer certaines branches explorées inutilement.



1.2 *Comparaison entre l'algorithme MiniMax et Alpha-Beta en termes de nœuds explorée*

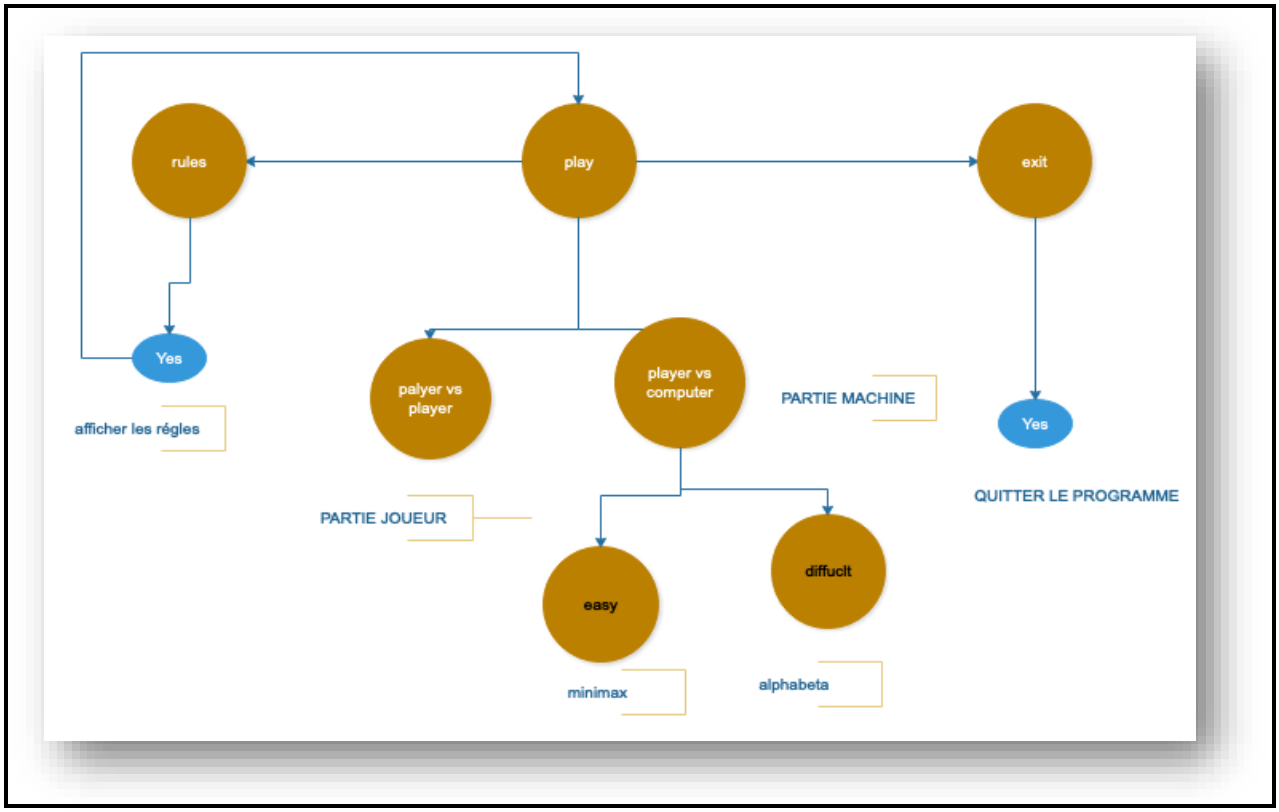
```
TYPE YOUR CHOICE HERE >>>> 1
le nbre de noueds generees est  0
 le nbre de noueds generees est  267744
 le nbre de noueds generees est  225355
 le nbre de noueds generees est  190060
 le nbre de noueds generees est  160649
 le nbre de noueds generees est  161187
 le nbre de noueds generees est  107561
 le nbre de noueds generees est  83900
```

```
              2 > AlphaBeta
TYPE YOUR CHOICE HERE >>>> 2
le nbre de noueds generees est  0
 le nbre de noueds generees est  111210
 le nbre de noueds generees est  136974
 le nbre de noueds generees est  113525
 le nbre de noueds generees est  70554
 le nbre de noueds generees est  124906
 le nbre de noueds generees est  61186
 le nbre de noueds generees est  77439
 le nbre de noueds generees est  45721
 le nbre de noueds generees est  43240
```

III. Méthodologie du travail

1.1 Analyse & conception

On a donné au joueur le choix entre une **partie joueur** ou une **partie machine**(IA). la version console est réalisée avec les deux stratégies MiniMax et Alpha-Beta ainsi que la version graphique est effectuée avec la stratégie Alpha-Beta (puisqu'il améliore l'algorithme MiniMax).



1.2 Développement

Puisque la console néglige un peu le côté ergonomique et esthétique demandés pour la réalisation des jeux .On a essayé d'améliorer la conception pour être bien adaptée aux conditions de travail du joueur et lui donner l'impression d'être effectivement dans une application jeu .Alors on a utilisé des fonctions pour ajouter les couleurs et aussi l'animation .

```c
void my_delay(int i)     /*Pause l'application pour i seconds*/
{
    clock_t start,end;
    start=clock();
    while(((end=clock())-start)<=(i*CLOCKS_PER_SEC)/5);
}

void color(int t,int f)
{
    HANDLE H=GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(H,f*16+t);
}
```

IV. Réalisation du projet

1.1 Interface console

- **Contenu du main :**

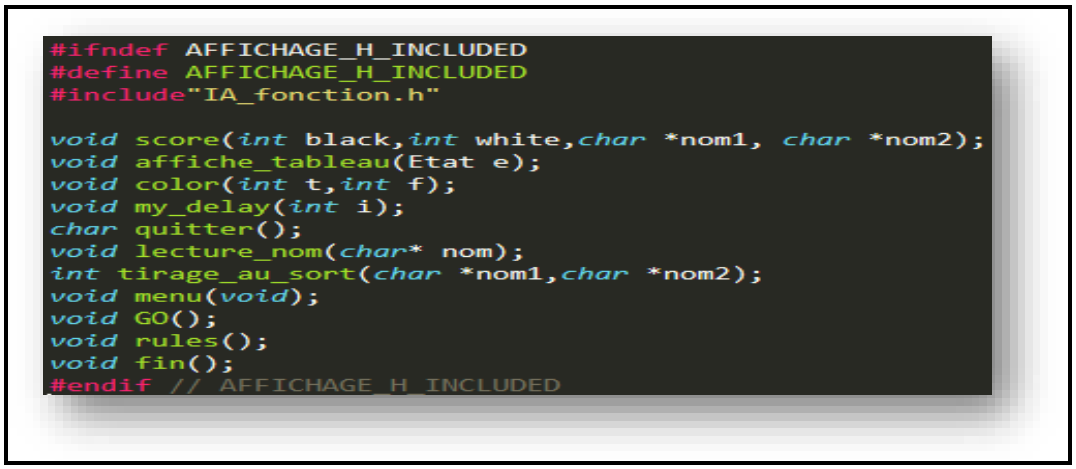
```c
#include <string.h>
#include "affichage.h"
#include "capture.h"
#include"IA_fonction.h"
#include "partie_joueur.h"
#include "partie_machine.h"
char nom1[10],nom2[10];
//kkkkkkkkkkkkk
int main(){
   int choix=1;
//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§ AFFICHAGE DU MENU §§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§

    l: GO();
       printf("\n\n\t\t1 > PLAY \n\n"); my_delay(4);
       printf("\t\t\t2 > GAME RULES\n\n"); my_delay(5);
       printf("\t\t\t\t3 > EXIT \n\n"); my_delay(5);
       printf("\nTYPE YOUR CHOICE HERE >>>> "); scanf("%d",&choix);getchar();
       system("cls");

//§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§ SWITCH §§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§§
```

```
    switch(choix)
        {    case 1: GO();
                     printf("\n\n\t1 > PLAYER Vs PLAYER \n\n"); my_delay(5);
                     printf("\t\t 2 > PLAYER Vs COMPUTER \n\n"); my_delay(4);
                     printf("\nTYPE YOUR CHOICE HERE >>>> "); scanf("%d",&choix);getchar();
                     if(choix==1)
                     {
                          printf("\n PLAYER 1:\n");
                               lecture_nom(nom1);
                               printf("PLAYER 2:\n");
                               lecture_nom(nom2);
                               partie_joueur(nom1,nom2);
                     }
                     if(choix==2)
                           partie_machine();
                     break;
             case 2: /* afficher les regles du jeu*/
                     rules();
                    goto 1;
                    break;
             case 3:  /* quitter nous donne le choix entre sortir ou continuer*/
                    GO();
                    if(quitter()=='y') return 0;
                    else goto 1;
                    break;
             default: GO();
                     printf("YOUR CHOICE DOES NOT EXIST !");
        }
ystem("pause");
eturn 0;
    }
```

- **Fonctions utilisées**

On a 4 fichiers Headers :

- **affichage.h :** qui contient les fonctions pour la mise en page du jeu comme les couleurs l'affichage du tableau, affichage des noms ...



```
#ifndef AFFICHAGE_H_INCLUDED
#define AFFICHAGE_H_INCLUDED
#include"IA_fonction.h"

void score(int black,int white,char *nom1, char *nom2);
void affiche_tableau(Etat e);
void color(int t,int f);
void my_delay(int i);
char quitter();
void lecture_nom(char* nom);
int tirage_au_sort(char *nom1,char *nom2);
void menu(void);
void GO();
void rules();
void fin();
#endif // AFFICHAGE_H_INCLUDED
```

- **partie_joueur.h :** qui contient les fonctions pour effectuer une partie joueur contre joueur.

```
#ifndef PARTIE_JOUEUR_H_INCLUDED
#define PARTIE_JOUEUR_H_INCLUDED



void partie_joueur(char *nom1,char *nom2);

int test_glisser(int posi_a,int posj_a,int posi_n,int posj_n);

int est_un_moulin(int posi,int posj,int br,Etat e);
#endif // PARTIE_JOUEUR_H_INCLUDED
```

- **partie_machine.h :** qui contient les fonctions suivantes pour effectuer une partie joueur contre machine.

```
#ifndef PARTIE_MACHINE_H_INCLUDED
#define PARTIE_MACHINE_H_INCLUDED

void partie_machine();
void machine_minimax(char* nom1,char *nom2,Etat etatinitial);
int minimax(Etat e, int p,int maxplayer);
void machine_alphabeta(char* nom1,char *nom2,Etat etatinitial);
int alphabeta(Etat e, int p,int a,int b,int maxplayer);
#endif // PARTIE_MACHINE_H_INCLUDED
```

- **IA_fonction.h :** qui contient les fonctions de bases pour la recherche des nœuds vu dans le cours, ainsi que les règles du jeu, la fonction heuristique.

```
void f_etatinitial(Etat *e);
liste* generesuccesseur(Etat etatcourant,int maxplayer,liste *l);
liste* glisser(Etat etatcourant,int maxplayer, liste *l );
liste* poser(Etat etatcourant,int maxplayer, liste *l );
liste* sauter(Etat etatcourant,int maxplayer, liste *l );
liste* enlevercase(Etat etatcourant,int maxplayer,liste *l);
//Etat enlever_case(Etat e,int maxplayer);
void inserer(Etat etatinitial,liste** listenoeud);
Etat extrairealphabeta(liste* l,int maxplayer);
Etat extraireminimax(liste* l,int maxplayer);
int etatsolution(Etat );
int h(Etat);
int nombre_coup(Etat e,int maxplayer);
int nombre_moulin(Etat e, int maxplayer);
void espace_prefere(Etat e, int maxplayer, int* nombre_moulin,int *moulinincomplet,int* x);
int utilitee(Etat);
```

Les fonctions de base :

- **generesuccesseurs :** génération des successeurs à partir d'un noeud en suivant les règles du jeu.

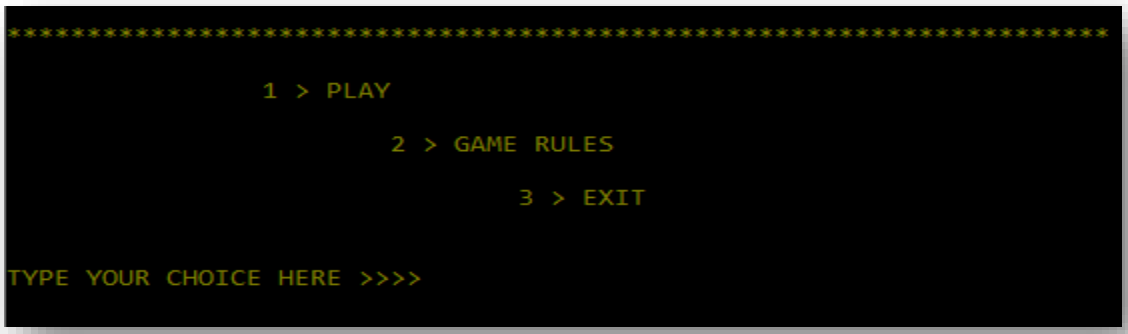
- **inserer :** Insertion d'un nouveau nœud dans la liste.

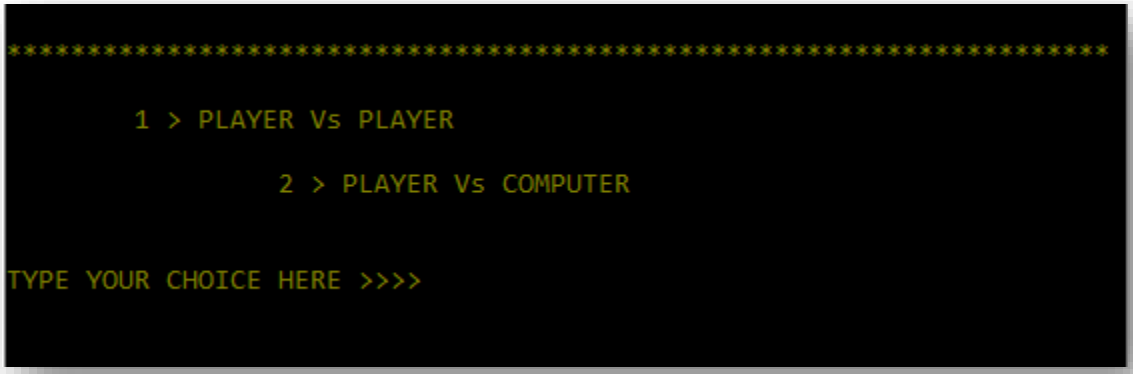
- **extraireminimax :** extraire un état de la liste en prenant en considération la valeur de la fonction heuristique h et fait appel à la fonction minimax.

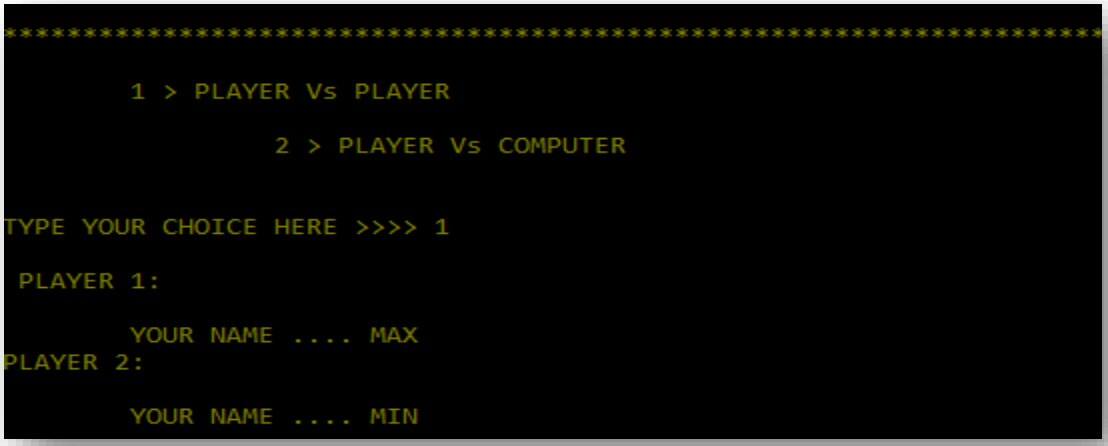
- **extrairealphabeta :** permet aussi l'extraction mais en faisant appel à la fonction alphabeta .

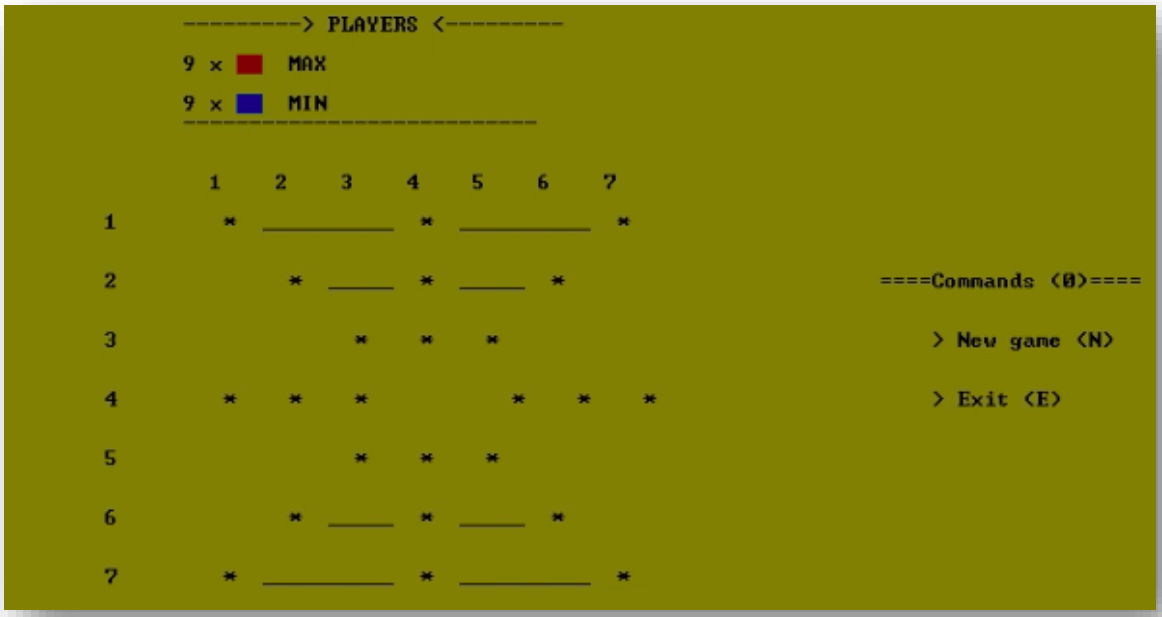
- **utilitee :** retourne 1000 si le joueur Max est gagnant, sinon -1000.

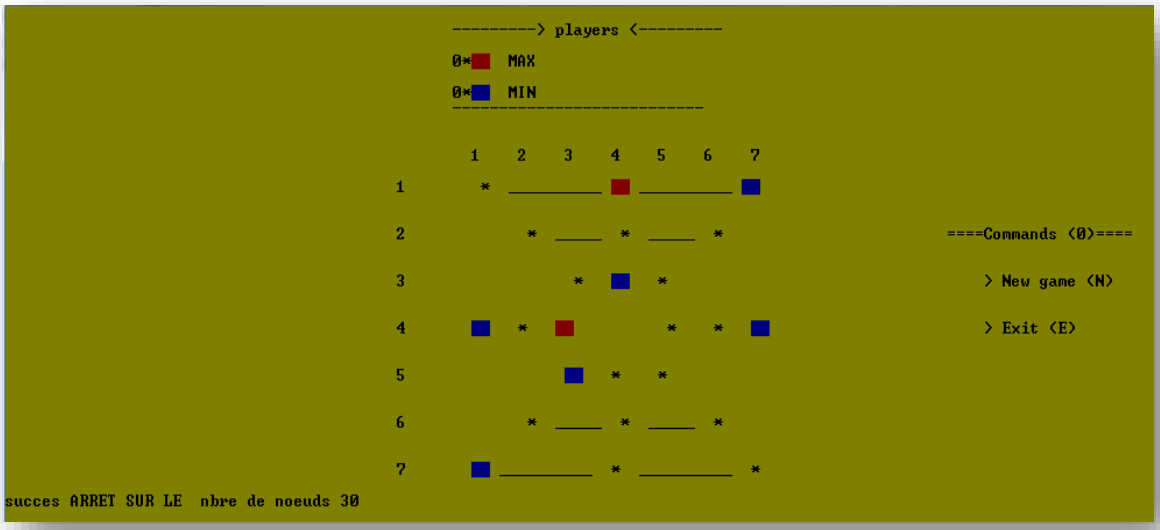
- **Exécution de la version console**








---------> PLAYERS <---------
9 x MAX
9 x MIN
1 2 3 4 5 6 7
1
2
3
4
5
6
7
====Commands (0)====
> New game (N)
> Exit (E)


---------> players <---------
0x MAX
0x MIN
1 2 3 4 5 6 7
1
2
3
4
5
6
7
====Commands (0)====
> New game (N)
> Exit (E)
succes ARRET SUR LE nbre de noeuds 30

```
**********************************************************************

        1 > PLAYER Vs PLAYER

                2 > PLAYER Vs COMPUTER


TYPE YOUR CHOICE HERE >>>> 2
```

```
**********************************************************************

        1 >MINIMAX
                2 > AlphaBeta
TYPE YOUR CHOICE HERE >>>> 1
```

La stratégie MiniMax commence par :

```
le nbre de noueds generees est  267744
```

```
**********************************************************************

        1 >MINIMAX
                2 > AlphaBeta
TYPE YOUR CHOICE HERE >>>> 2
```

La stratégie Alpha-Beta commence par :

```
le nbre de noueds generees est  111210
```

1.2 *Interface graphique*

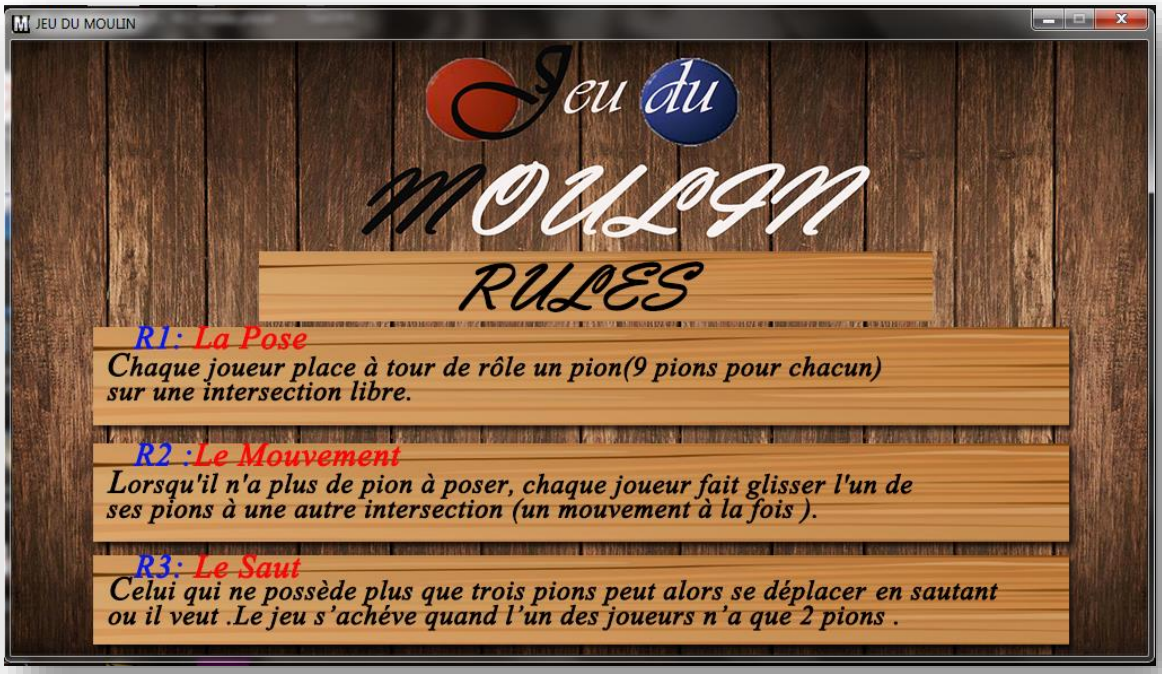
L'exécution du programme est comme suit :

- Le jeu s'affiche avec une première page qui ne dure pas assez de temps(1,5 s) :

- Ensuite le menu du jeu s'affiche :

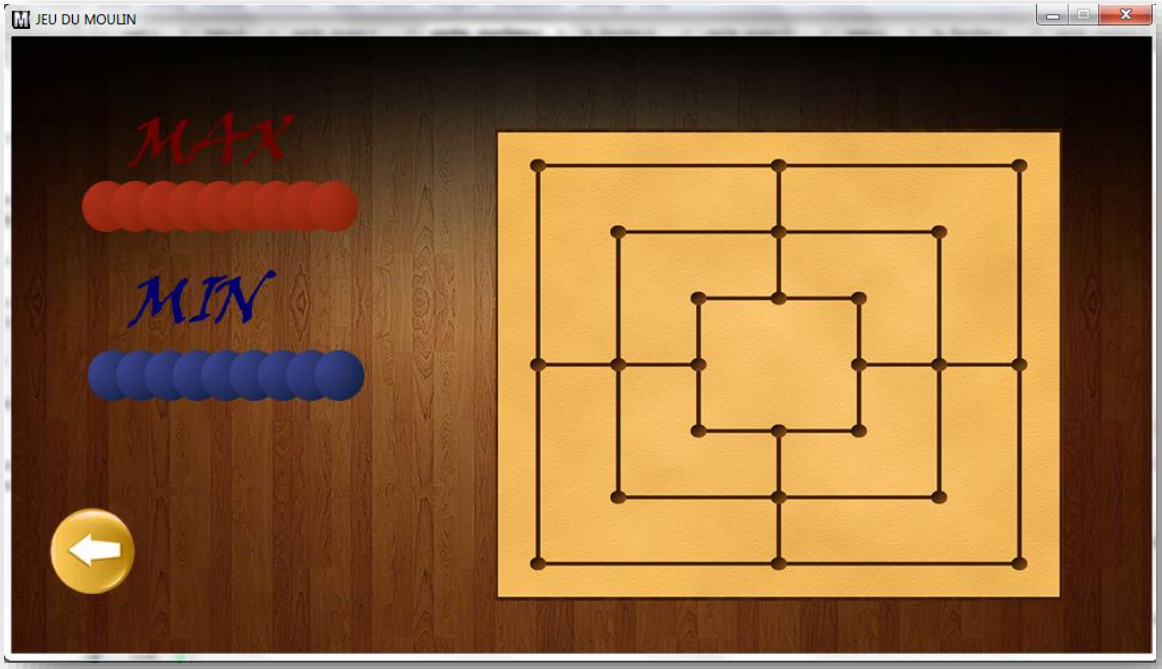
- Si on la souris se déplace tout en long du menu les choix changent :

- Si on clique sur Rules l'image suivante s'affiche :

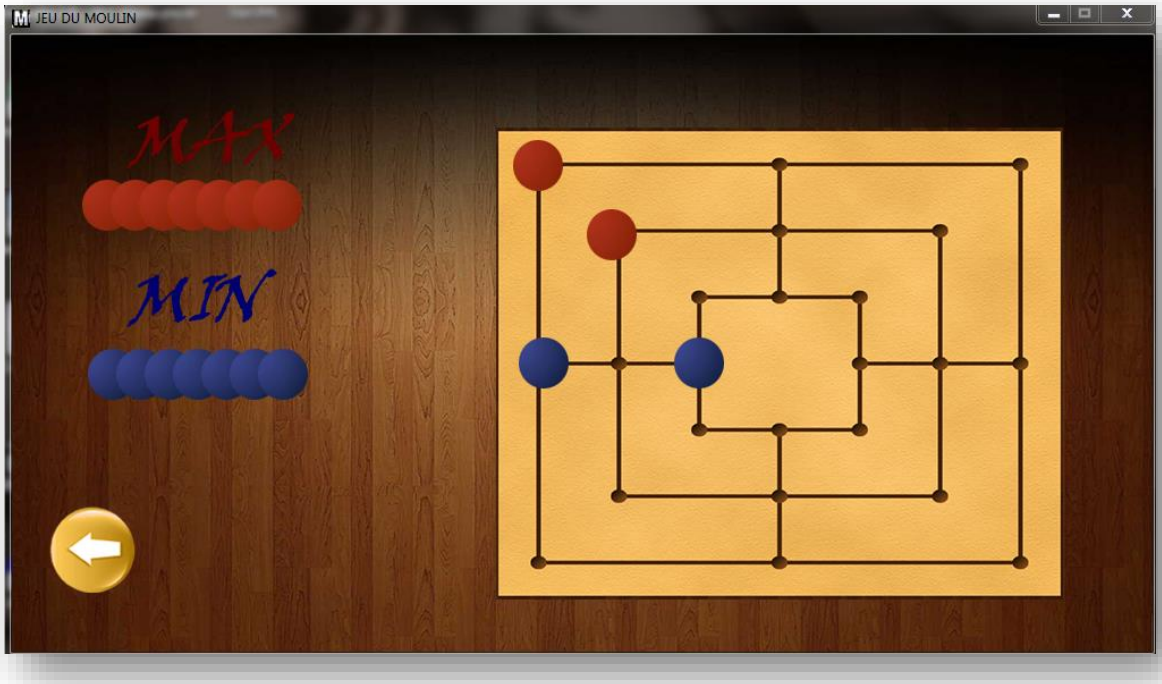


- En cliquant sur le choix Play on obtient :

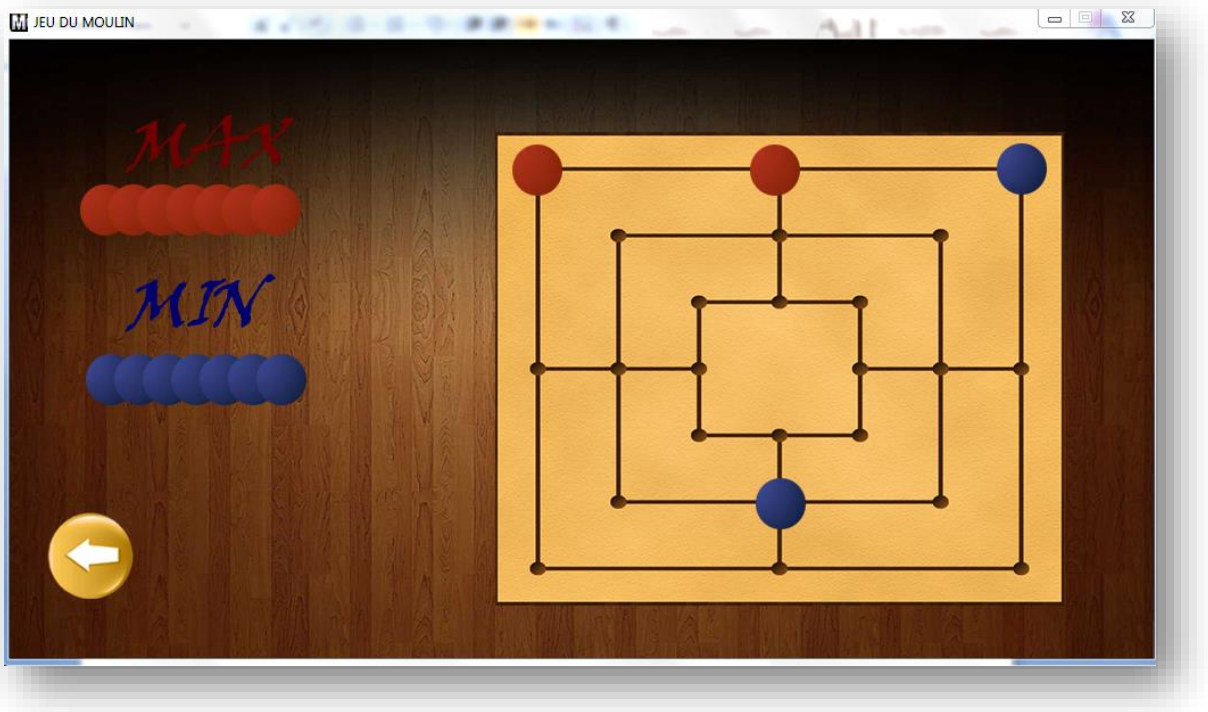
- Le plateau du jeu et les noms des joueurs sont affichés comme suit :



- Si on a le cas de joueur Vs joueur :



- Si on a le cas de joueur Vs machine tels que Max est le joueur et Min représente la machine :



V. Conclusion

Ce projet fut une très bonne expérience, vu qu'il a testé nos connaissances acquises lors du module résolution des problèmes dans ce 2ème semestre. On a également pu améliorer nos compétences en modélisation et développement des différents algorithmes de recherche ainsi il nous a permet aussi de se familiariser avec la bibliothèque graphique SDL.